

HotSpot Just-In-Time Compilers

TriView: Source, Bytecode, Assembly Viewer

Class: `org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog` Member: `private long add(long, long)`

☒ Source ☒ Bytecode ☒ Assembly

Bytecode size: `4` Native size: `88` Compile time (ms): `4`

Source	Bytecode (double click for JVMs)	Assembly
157 <code>sum = Math.min(i, i+1);</code>	0: <code>lload_1</code>	0x00007f4760710340: <code>cmp 0x8(%rsi),%rax</code>
158 <code>}</code>	1: <code>lload_3</code>	0x00007f4760710344: <code>jne 0x00007f47606cf960</code>
159 <code>System.out.println("intrinsicTest:</code>	2: <code>ladd</code>	0x00007f476071034a: <code>xchg %ax,%ax</code>
160 <code>}</code>	3: <code>lreturn</code>	0x00007f476071034c: <code>nopl 0x0(%rax)</code>
161 <code>}</code>		[Verified Entry Point]
162 <code>private long add(long a, long b)</code>		0x00007f4760710350: <code>sub \$0x18,%rsp</code>
163 <code>{</code>		0x00007f4760710357: <code>mov %rbp,0x10(%rsp) ;*s</code>
164 <code>return a + b;</code>		0x00007f476071035c: <code>mov %rdx,%rax</code>
165 <code>{</code>		0x00007f476071035f: <code>add %rcx,%rax ;*ladd</code>
166 <code>return a + b;</code>		0x00007f4760710362: <code>add \$0x10,%rsp</code>
167 <code>}</code>		0x00007f4760710366: <code>pop %rbp</code>
168 <code>private long sub(long a, long b)</code>		0x00007f4760710367: <code>test %eax,0x5cedc93(%rip)</code>
169 <code>{</code>		0x00007f476071036d: <code>retq</code>
170 <code>return a - b;</code>		0x00007f476071036e: <code>hlt</code>
171 <code>}</code>		0x00007f476071036f: <code>hlt</code>
172 <code>public void tooBigToInline(int iterations)</code>		0x00007f4760710370: <code>hlt</code>
173 <code>{</code>		0x00007f4760710371: <code>hlt</code>
174 <code>long count = 0;</code>		0x00007f4760710372: <code>hlt</code>
175 <code>for (int i = 0; i < iterations; i++)</code>		0x00007f4760710373: <code>hlt</code>
176 <code>{</code>		0x00007f4760710374: <code>hlt</code>
177 <code>count = bigMethod(count, i);</code>		0x00007f4760710375: <code>hlt</code>
178 <code>}</code>		0x00007f4760710376: <code>hlt</code>
179 <code>System.out.println("tooBigToInline:</code>		0x00007f4760710377: <code>hlt</code>
180 <code>}</code>		0x00007f4760710378: <code>hlt</code>
181 <code>Mounted class version: 51.0 (Java 7) private long add(long, long) compiled with C2</code>		0x00007f4760710379: <code>hlt</code>
		0x00007f476071037a: <code>hlt</code>

WhatSpot?

- Java HotSpot Virtual Machine
 - Bytecode-interpreting stack machine
 - No registers
 - Variables pushed onto stack
 - Just In Time (JIT) compilers
 - Profile Guided Optimisation (PGO)
 - Compile bytecode to native code

Tiered	Non-Tiered	-Xint	-Xcomp
2.9s	2.6s	80.5s	4.4s

*Horrible unscientific benchmark
(com.chrisnewland.jitwatch.demo.MakeHotSpotLog)

Talking JIT

- Client compiler (C1)
 - Starts quickly, compilation to native, inlining
- Server compiler (C2)
 - Waits until more information available
 - Loop unrolling, **Inlining**, Dead Code Elimination, Escape analysis, Intrinsics, **Branch prediction**
- Tiered Compilation (C1 + C2)
 - Default in Java 8
 - Enable in Java 7 with -XX:+TieredCompilation
 - Best of both worlds?

Explain yourself!

- Enable JIT logging
- -XX:+UnlockDiagnosticVMOptions
- -XX:+LogCompilation
- -XX:+TraceClassLoading (JITWatch)
- -XX:+PrintAssembly
 - Requires hsdis binary in jre/lib/<arch>/server
 - Significant performance overhead
 - <http://www.chrisnewland.com/building-hsdis-on-linux-amd64-on-debian-369>
 - <http://www.chrisnewland.com/building-hsdis-amd64dylib-on-mac-osx-376>

I heard you like to grep?

```
<task compile_id='23' method='java/util/ArrayList$Itr
checkForComodification ()V' bytes='23' count='9006'
backedge_count='1' iicount='44000' st
amp='1.603'>
<phase name='parse' nodes='3' live='3' stamp='1.603'>
<type id='680' name='void' />
<klass id='776' name='java/util/ArrayList$Itr' flags='2' />
<method id='777' holder='776' name='checkForComodification'
return='680' flags='16' bytes='23' iicount='44000' />
<klass id='781' name='java/util/ConcurrentModificationException'
unloaded='1' />
<uncommon_trap method='777' bci='14' reason='unloaded'
action='reinterpret' index='47' klass='781' />
<parse method='777' uses='44000' stamp='1.604'>
<bc code='180' bci='4' />
...
```

- Logs can be > 50MB of XML :(
- Much bigger with disassembly!
- Let's build a visualiser!

JITWatch

- <https://github.com/AdoptOpenJDK/jitwatch/>
- JIT Compilation
 - When? (time, invocations)
 - How? (C1, C2, Tiered, OSR)
- Decompiles
 - Back to bytecode interpretation (Why?)
- Inlining successes and failures
- Branch probabilities - taken / not taken
- Intrinsic

Inlining (C1 + C2)

```
int a = 3;  
int b = 4;  
int result = add(a, b);  
...  
public int add(int x, int y) { return x + y; }
```



```
int result = a + b;
```

Inlined only if bytecode size is < 35 bytes or <325 bytes and code is hot*

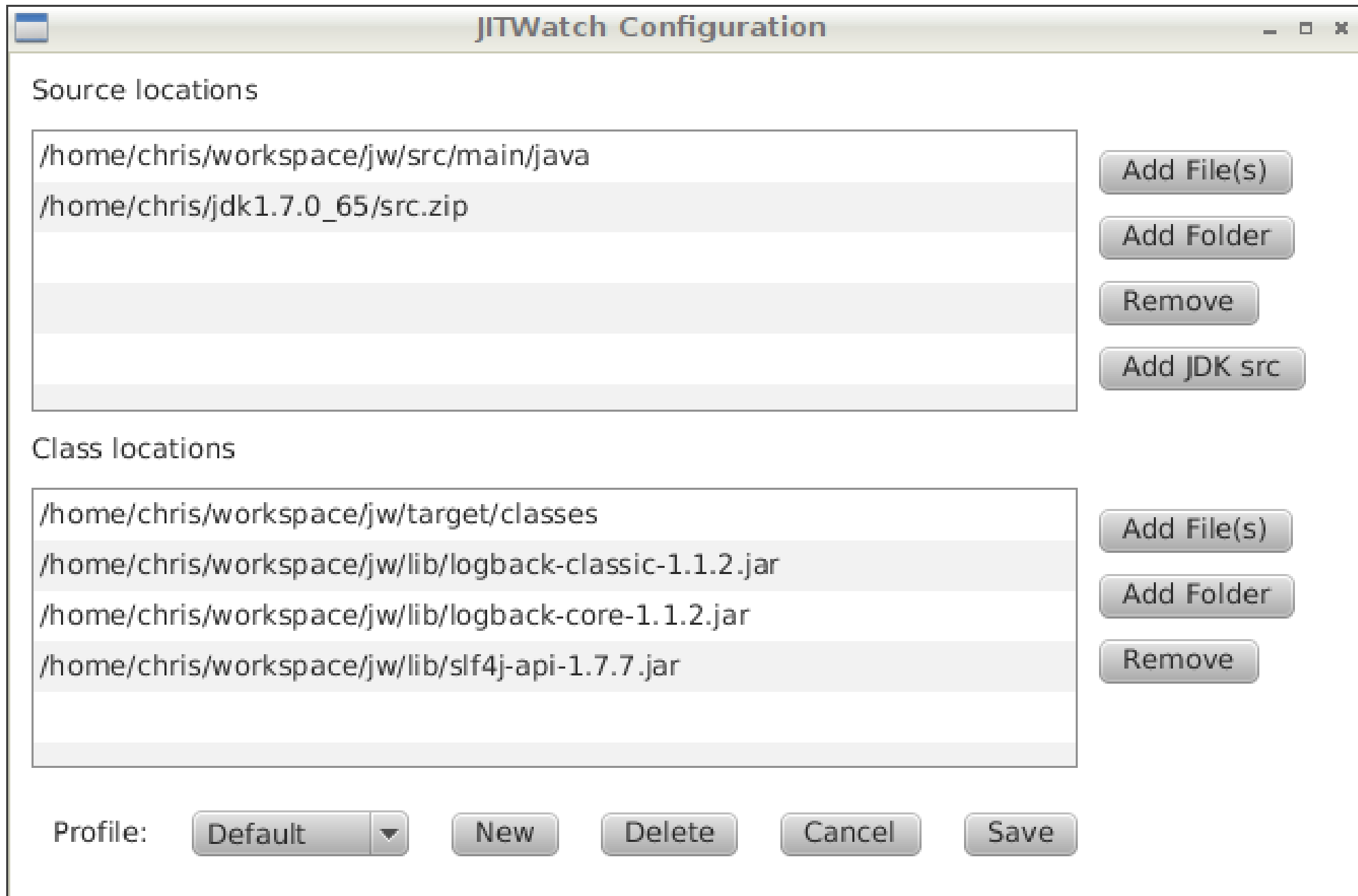
Branch Prediction (C2)

```
// make an array of random doubles 0..1
double[] bigArray = makeBigArray(1_000_000);

for (int i = 0; i < bigArray.length; i++)
{
    double cur = bigArray[i];
    if (cur > 0.5) { doThis(); } else { doThat(); }
}

// branch will be taken ~50% of time
// sorting the array will make it more predictable
```


Setting up



The image shows a 'JITWatch Configuration' dialog box. It has a title bar with a blue icon, the text 'JITWatch Configuration', and standard window controls. The dialog is divided into two main sections: 'Source locations' and 'Class locations'. Each section has a list box on the left and a set of buttons on the right. The 'Source locations' list contains two entries: '/home/chris/workspace/jw/src/main/java' and '/home/chris/jdk1.7.0_65/src.zip'. The 'Class locations' list contains four entries: '/home/chris/workspace/jw/target/classes', '/home/chris/workspace/jw/lib/logback-classic-1.1.2.jar', '/home/chris/workspace/jw/lib/logback-core-1.1.2.jar', and '/home/chris/workspace/jw/lib/slf4j-api-1.7.7.jar'. At the bottom, there is a 'Profile:' label, a dropdown menu showing 'Default', and four buttons: 'New', 'Delete', 'Cancel', and 'Save'.

JITWatch Configuration

Source locations

- /home/chris/workspace/jw/src/main/java
- /home/chris/jdk1.7.0_65/src.zip

Class locations

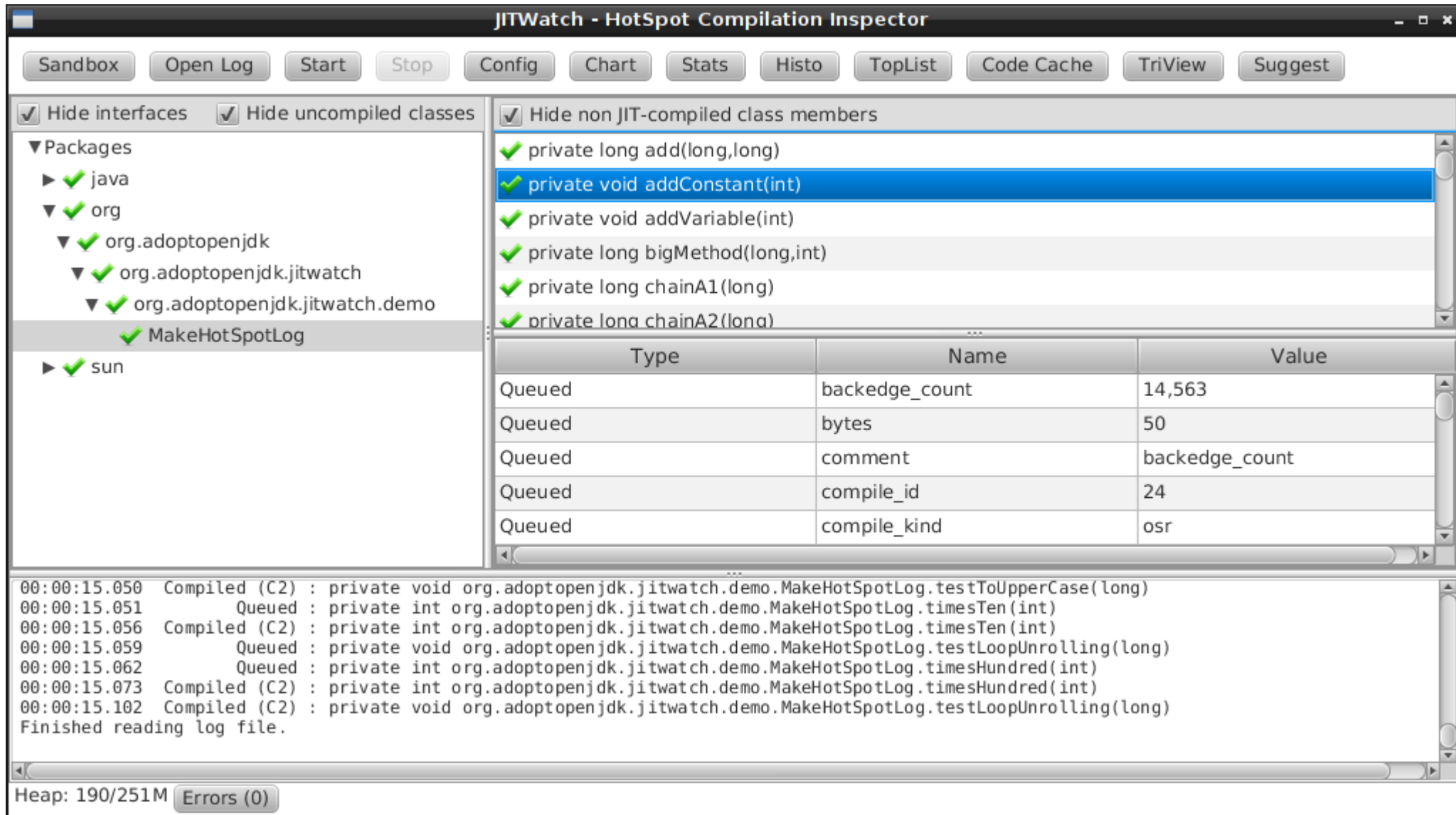
- /home/chris/workspace/jw/target/classes
- /home/chris/workspace/jw/lib/logback-classic-1.1.2.jar
- /home/chris/workspace/jw/lib/logback-core-1.1.2.jar
- /home/chris/workspace/jw/lib/slf4j-api-1.7.7.jar

Profile: **Default** ▼ **New** **Delete** **Cancel** **Save**

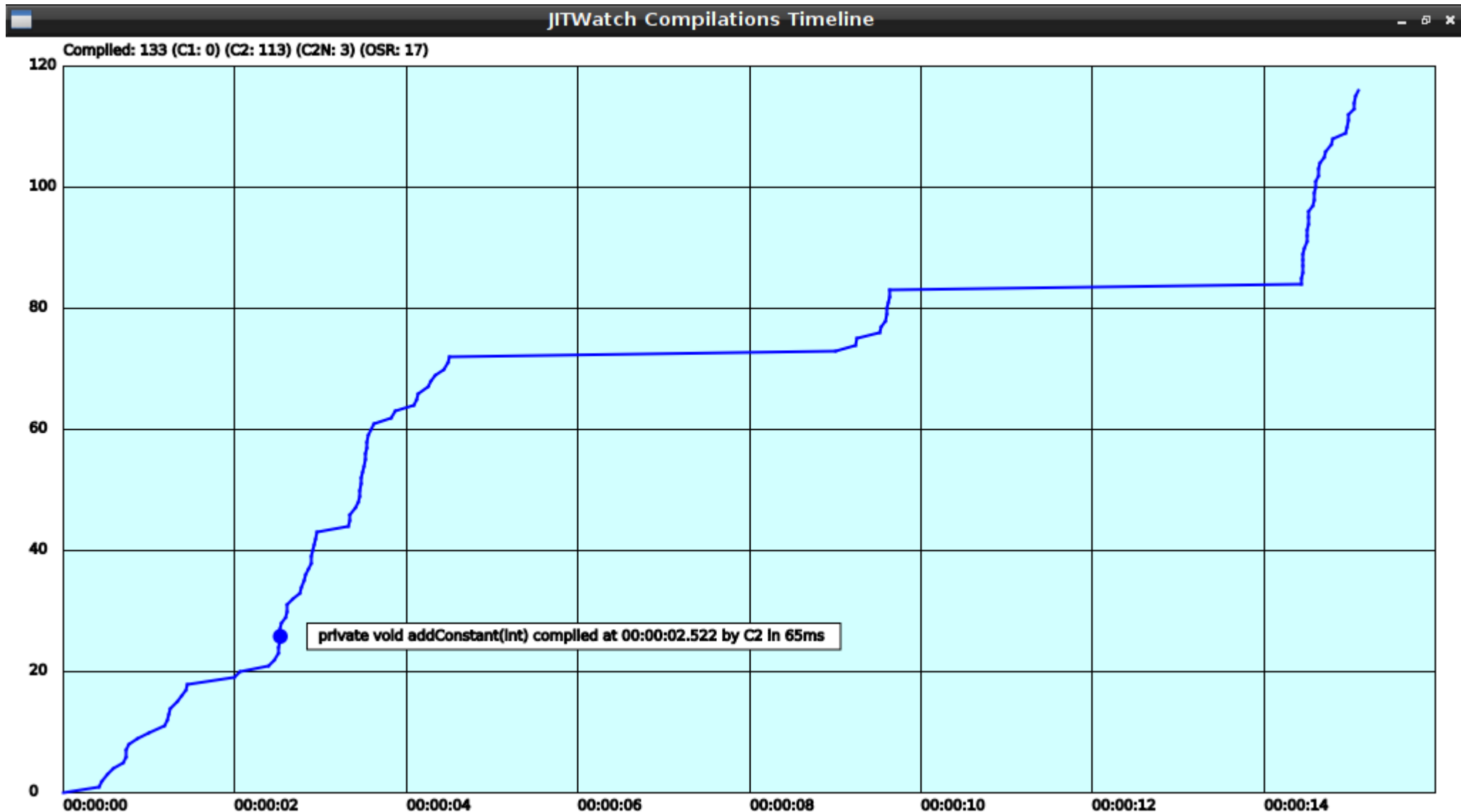
Buttons on the right:

- Add File(s)
- Add Folder
- Remove
- Add JDK src
- Add File(s)
- Add Folder
- Remove

Compile tree



Compilations timeline



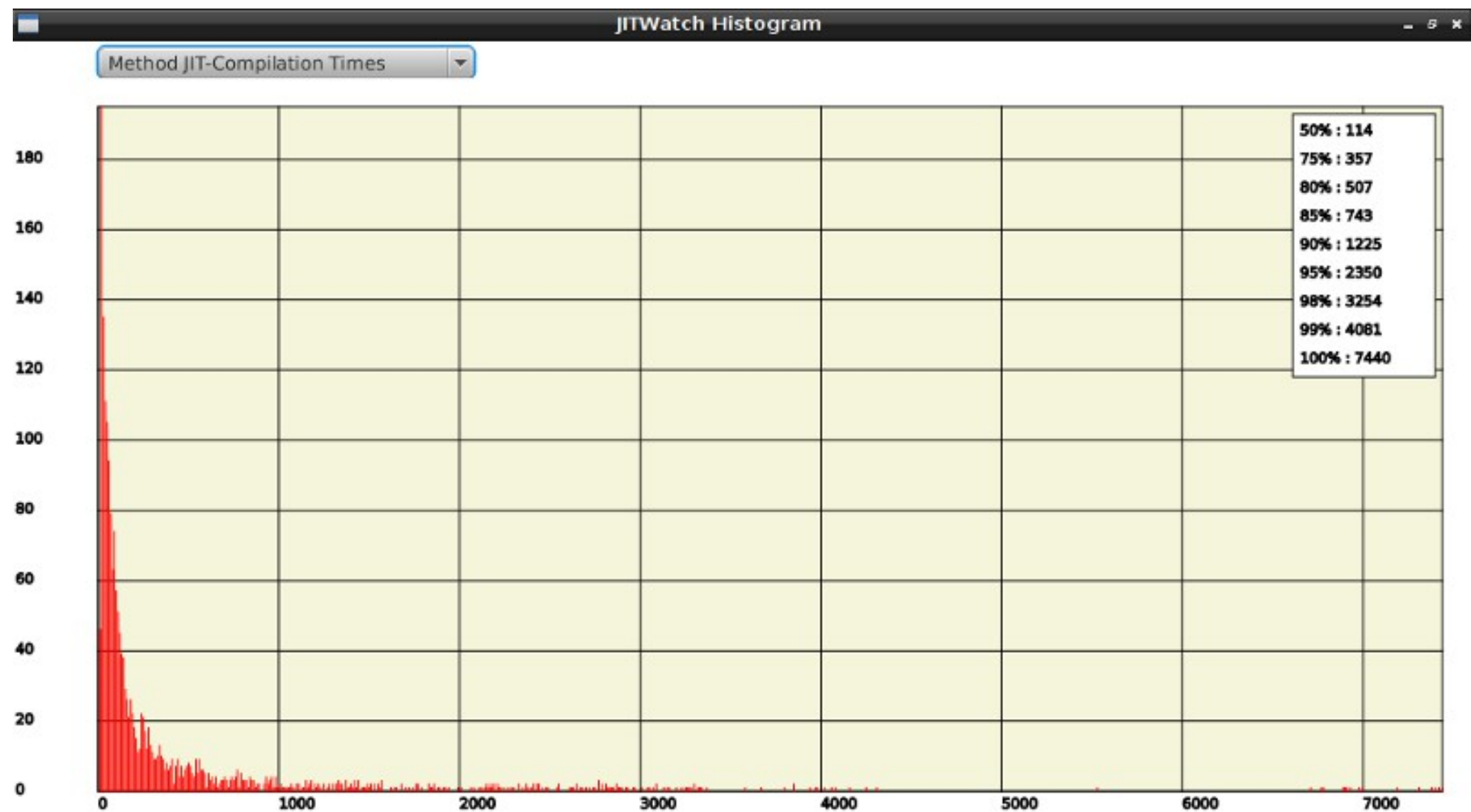
Toplists

- **Bytecode size**
- Native code size
- **Inlining failure reasons**
- Most-used intrinsics
- Compilation order
- Most-decompiled methods
 - Compiler assumption was wrong

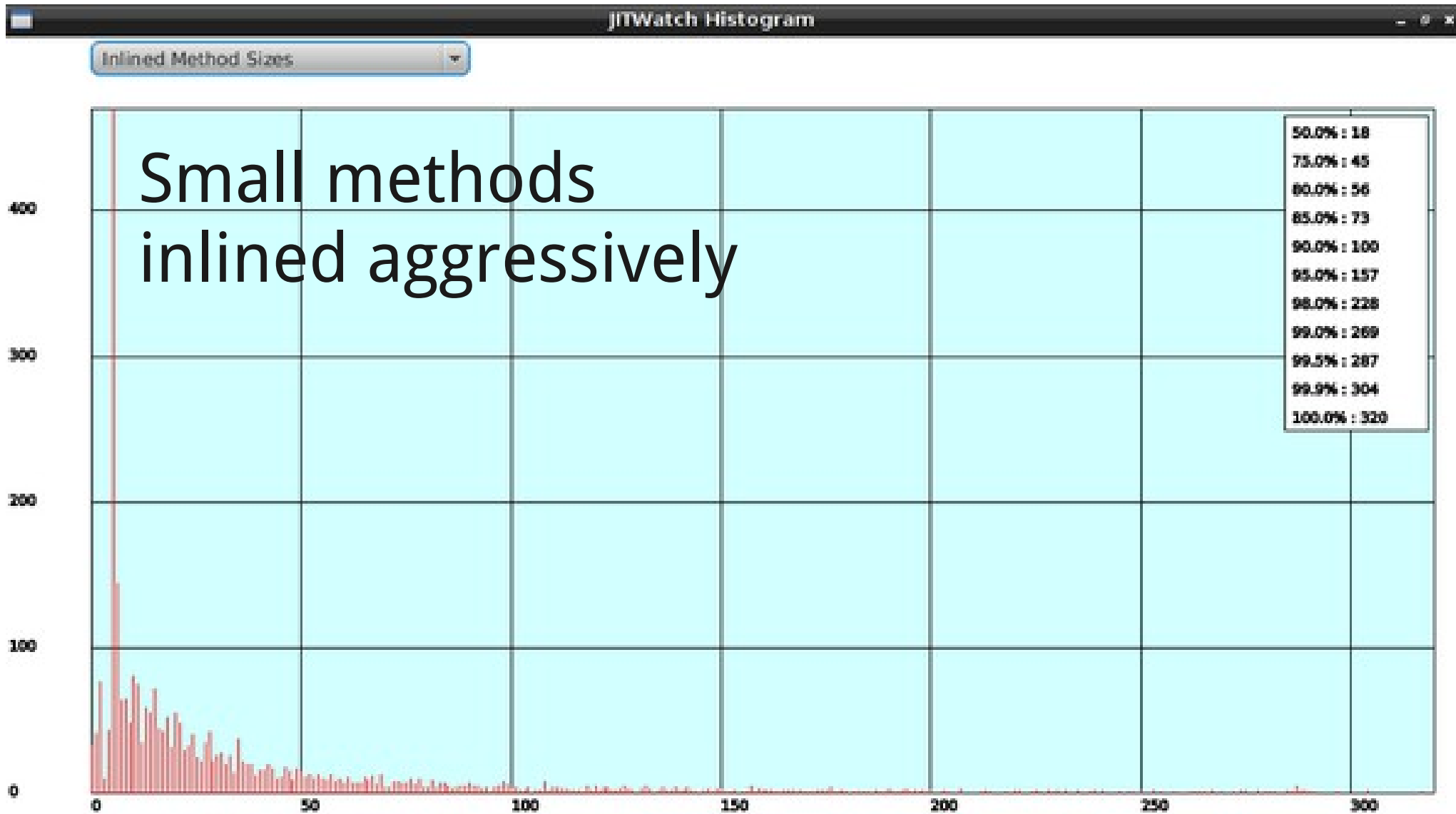
Toplists – Inline failure reasons

[illegible]

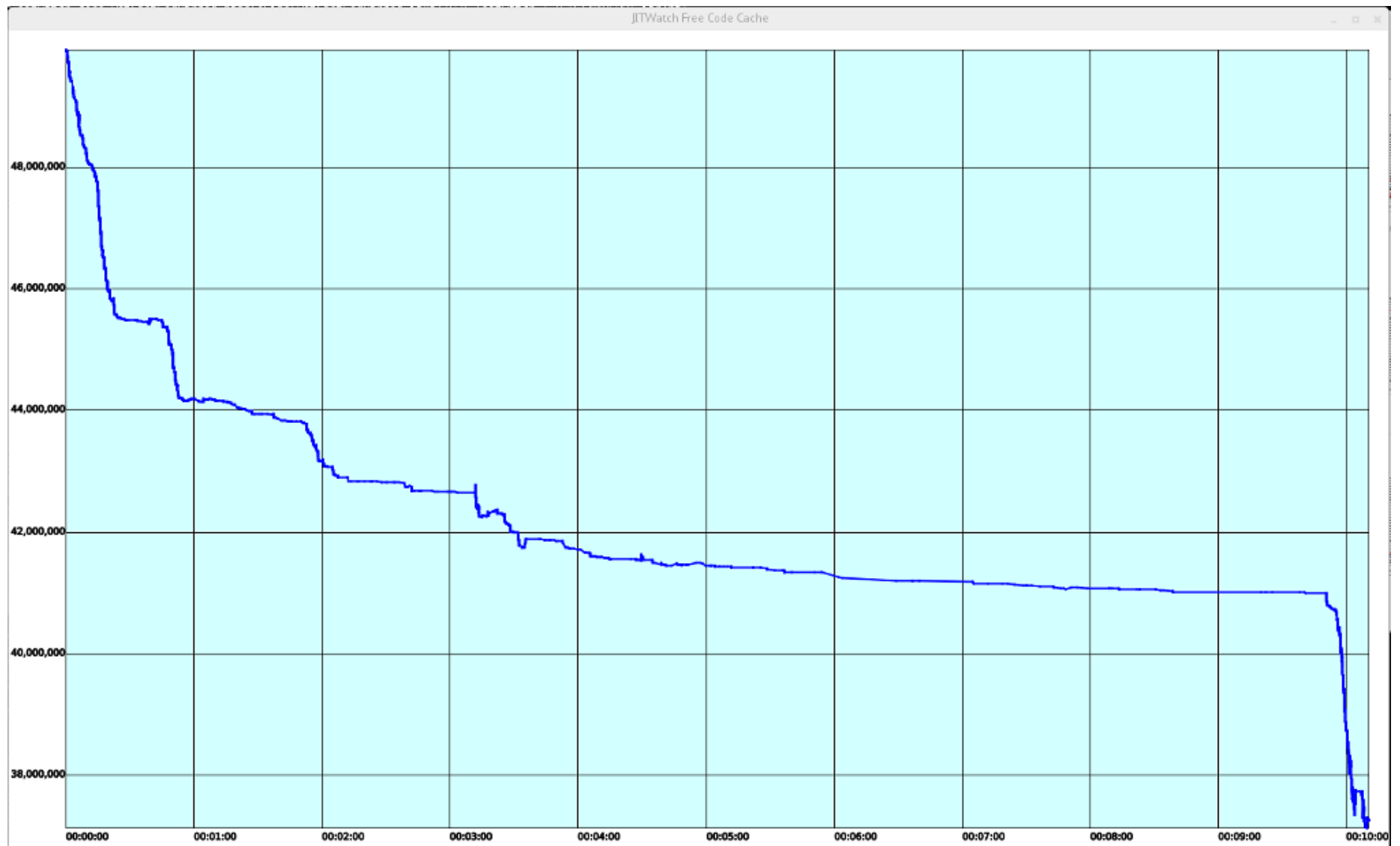
Compile times



Histogram - Inlined method sizes



Code Cache



TriView

TriView: Source, Bytecode, Assembly Viewer

Class: Member:

☒ Source ☒ Bytecode ☒ Assembly

	Bytecode size	Native size	Compile time (ms)
	4	88	4

Source	Bytecode (double click for JVMs)	Assembly
157 sum = Math.min(i, i+1);	0: lload 1	0x00007f4760710340: cmp 0x8(%rsi),%rax
158 }	1: lload 3	0x00007f4760710344: jne 0x00007f47606cf960
159	2: ladd	0x00007f476071034a: xchg %ax,%ax
160 System.out.println("intrinsicTest:	3: lreturn	0x00007f476071034c: nopl 0x0(%rax)
161 }		[Verified Entry Point]
162		0x00007f4760710350: sub \$0x18,%rsp
163		0x00007f4760710357: mov %rbp,0x10(%rsp) ;*s
164 private long add(long a, long b)		;
165 {		0x00007f476071035c: mov %rdx,%rax
166 return a + b;		0x00007f476071035f: add %rcx,%rax ;*ladd
167 }		;
168		0x00007f4760710362: add \$0x10,%rsp
169 private long sub(long a, long b)		0x00007f4760710366: pop %rbp
170 {		0x00007f4760710367: test %eax,0x5cedc93(%rip)
171 return a - b;		
172 }		0x00007f476071036d: retq
173		0x00007f476071036e: hlt
174 public void tooBigToInline(int iterat		0x00007f476071036f: hlt
175 {		0x00007f4760710370: hlt
176 long count = 0;		0x00007f4760710371: hlt
177		0x00007f4760710372: hlt
178 for (int i = 0; i < iterations; i++		0x00007f4760710373: hlt
179 {		0x00007f4760710374: hlt
180 count = bigMethod(count, i);		0x00007f4760710375: hlt
181 }		0x00007f4760710376: hlt
182		0x00007f4760710377: hlt
183 System.out.println("tooBigToInline:		0x00007f4760710378: hlt
184 }		0x00007f4760710379: hlt
185		0x00007f476071037a: hlt

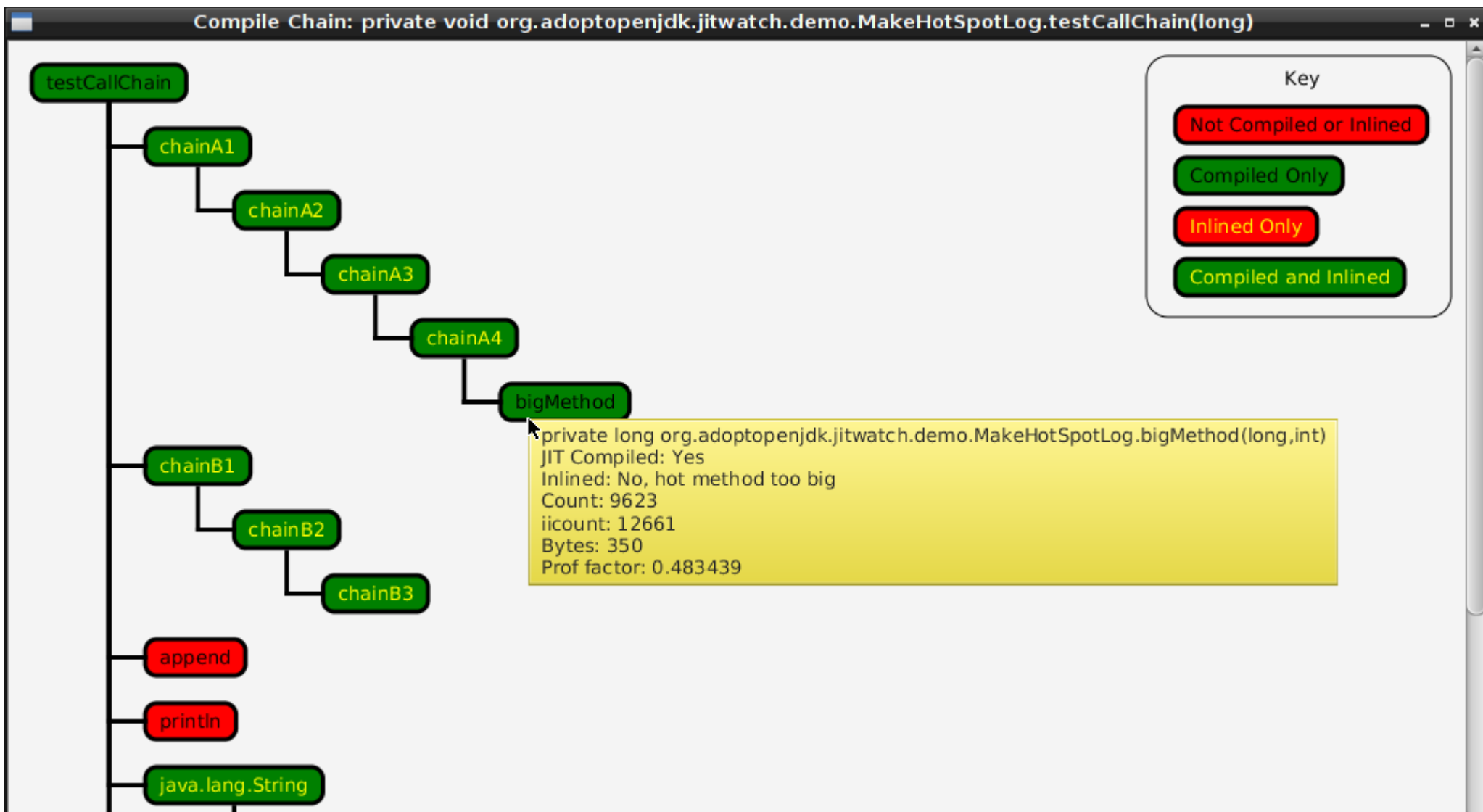
Mounted class version: 51.0 (Java 7) private long add(long,long) compiled with C2

JVM Spec Browser

The screenshot shows a web browser window titled "JMVSpec Browser - if_icmpge". The page content is organized into several sections:

- if_icmp<cond>**: The instruction name, displayed in a grey header bar.
- Operation**: Describes the instruction's function: "Branch if int comparison succeeds".
- Format**: Shows the instruction's byte layout: `if_icmp<cond>`, `branchbyte1`, and `branchbyte2`. This section is highlighted with a red border.
- Forms**: Lists the instruction forms and their corresponding opcode values in hexadecimal:
 - `if_icmpeq` = 159 (0x9f)
 - `if_icmpne` = 160 (0xa0)
 - `if_icmplt` = 161 (0xa1)
 - `if_icmpge` = 162 (0xa2)
 - `if_icmpgt` = 163 (0xa3)
 - `if_icmple` = 164 (0xa4)
- Operand Stack**: Shows the stack manipulation: `..., value1, value2 →`.

Compile Chains



Code Suggestion Tool

JITWatch Code Suggestions			
Score	Type	Caller	Suggestion
12668	Inlinling	org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog private long chainA4(long) View	The call at bytecode 3 to Class: org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog Member: private long bigMethod(long,int) was not inlined for reason: 'hot method too big' The callee method is 'hot' but is too big to be inlined into the caller. You may want to consider refactoring the callee into smaller methods. Invocations: 12668 Size of callee bytecode: 350
12668	Inlinling	org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog public void tooBigToInline(int) View	The call at bytecode 15 to Class: org.adoptopenjdk.jitwatch.demo.MakeHotSpotLog Member: private long bigMethod(long,int) was not inlined for reason: 'hot method too big' The callee method is 'hot' but is too big to be inlined into the caller. You may want to consider refactoring the callee into smaller methods. Invocations: 12668 Size of callee bytecode: 350
10076	Inlinling	java.util.ComparableTimSort private void mergeCollapse() View	The call at bytecode 69 to Class: java.util.ComparableTimSort Member: private void mergeAt(int) was not inlined for reason: 'hot method too big' The callee method is 'hot' but is too big to be inlined into the caller. You may want to consider refactoring the callee into smaller methods. Invocations: 10076 Size of callee bytecode: 356
10076	Inlinling	java.util.ComparableTimSort private void mergeCollapse() View	The call at bytecode 94 to Class: java.util.ComparableTimSort Member: private void mergeAt(int)

JMH support (perf annotations)

TriView: Source, Bytecode, Assembly Viewer

Class: `org.openjdk.jmh.samples.generated.JMHSample_08_DeadCo` Member: `public void measureRight_avgt_jmhLoop(InfraControl,...`

☐ Source ☐ Bytecode ☒ Assembly [View Compile Chain](#)

	Bytecode size	Native size	Compile time (ms)
	55	1360	11

Assembly

```
0x00007f25cd19c89f: vmovsd 0xa8(%r13),%xmm2 ; - org.openjdk.jmh.infra.Blackhole::consume@2 (line 386)
                                ; - org.openjdk.jmh.samples.generated.JMHSample_08_DeadCode_measureRight::me
                                ; *getfield d2
0x00007f25cd19c8a8: fldln2 ; - org.openjdk.jmh.infra.Blackhole::consume@16 (line 386)
0x00007f25cd19c8aa: sub $0x8,%rsp ; - org.openjdk.jmh.samples.generated.JMHSample_08_DeadCode_measureRight::me
0x00007f25cd19c8ae: vmovsd %xmm1, (%rsp)
0x00007f25cd19c8b3: fldl (%rsp)
0x00007f25cd19c8b6: fyl2x
0x00007f25cd19c8b8: fstpl (%rsp)
0x00007f25cd19c8bb: vmovsd (%rsp),%xmm1
0x00007f25cd19c8c0: add $0x8,%rsp ; *invokestatic
                                ; - org.openjdk.jmh.infra.Blackhole::consume@4 (line 71)
                                ; - org.openjdk.jmh.samples.generated.JMHSample_08_DeadCode_measureRight::measureRigh
0x00007f25cd19c8c4: vucomisd %xmm2,%xmm1
0x00007f25cd19c8c8: jp 0x00007f25cd19c8cc
0x00007f25cd19c8ca: je 0x00007f25cd19c908 ; *return
                                ; - org.openjdk.jmh.infra.Blackhole::consume@40 (line 390)
                                ; - org.openjdk.jmh.samples.generated.JMHSample_08_DeadCode_measureRight::mea
0x00007f25cd19c8cc: movzbl 0x94(%r14),%r10d ; *getfield isDone
                                ; - org.openjdk.jmh.samples.generated.JMHSample_08_DeadCode_measureRight::m
0x00007f25cd19c8d4: add $0x1,%rbp ; OopMap{r8=Oop rbx=Oop r13=Oop r14=Oop off=248}
                                ; *ifeq
                                ; - org.openjdk.jmh.samples.generated.JMHSample_08_DeadCode_measureRight::measureRigh
0x00007f25cd19c8d8: test %eax,0x17adf722(%rip) # 0x00007f25e4c7c000
```

Mounted class version: 50.0 (Java 6.0) public void measureRight_avgt_jmhLoop(InfraControl,RawResults,JMHSample_08_DeadCode_measureRight\$JMHSa...

Now with some Scala support! (thanks Ignasi!)

TriView: Source, Bytecode, Assembly Viewer

Class: Member:

☒ Source ☒ Bytecode ☒ Assembly

Bytecode size	Native size	Compile time (ms)
4	88	6

Source	Bytecode (double click for JVMs)	Assembly
1 class Hello { 2 def foo() = println("foo") 3 } 4 5 object Hello { 6 def main(args: Array[String]) = { 7 var i:Int=0 8 for(j <- (1 to 10000)) { i = inc(i) } 9 println(i); 10 } 11 12 def inc(i:Int):Int=i+1 13 }	0: iload 1 1: iconst 1 2: iadd 3: ireturn	0x00007f35a47b410a: xchg %ax,%ax 0x00007f35a47b410c: nopl 0x0(%rax) [Verified Entry Point] 0x00007f35a47b4110: sub \$0x18,%rsp 0x00007f35a47b4117: mov %rbp,0x10(%rsp) ;*synchronizatio ; - Hello\$::inc@ 0x00007f35a47b411c: mov %edx,%eax 0x00007f35a47b411e: inc %eax ;*iadd ; - Hello\$::inc@2 (line 12) 0x00007f35a47b4120: add \$0x10,%rsp 0x00007f35a47b4124: pop %rbp 0x00007f35a47b4125: test %eax,0x5cfaed5(%rip) # 0x00007f ; {poll_ 0x00007f35a47b412b: retq 0x00007f35a47b412c: hlt 0x00007f35a47b412d: hlt 0x00007f35a47b412e: hlt 0x00007f35a47b412f: hlt 0x00007f35a47b4130: hlt 0x00007f35a47b4131: hlt 0x00007f35a47b4132: hlt 0x00007f35a47b4133: hlt 0x00007f35a47b4134: hlt 0x00007f35a47b4135: hlt 0x00007f35a47b4136: hlt 0x00007f35a47b4137: hlt 0x00007f35a47b4138: hlt 0x00007f35a47b4139: hlt 0x00007f35a47b413a: hlt

Mounted class version: 50.0 (Java 6.0) public int inc(int) compiled with C2

JarScan Tool

- Static analysis of a jar
- Finds bytecode > inlining threshold
- These methods might not be hot
- Around 3000 non-inlineable methods in rt.jar
 - String.split
 - String.toUpperCase / toLowerCase
 - Core parts of j.u.ComparableTimSort

Be assertive (or don't)

```
// java.util.Collections

public static <T extends Comparable<? super T>>
void sort(List<T> list)
{
    Object[] a = list.toArray();
    Arrays.sort(a);
    ...
}
```


Too big to inline

```
// java.util.Arrays.java
```

```
public static void sort(Object[] a)
{
    ...
    ComparableTimSort.sort(a, 0, a.length,
null, 0, 0);
}
```

```
327 -> private static int gallopLeft(...)
```

```
716 -> private void mergeHi(...)
```

```
652 -> private void mergeLo(...)
```

```
327 -> private static int gallopRight(...)
```

Assert no more

Bench	Mode	Samples	Score	error
testSort	avgt	10	2957.302	45.569
testSortNoAssertions	avgt	10	3046.972	98.393

gallopLeft 327 → 244 (-25.4%)

gallopRight 327 → 244 (-25.4%)

mergeHi 716 → 583 (-18.6%)

mergeLo 652 → 517 (-20.7%)

Other inlining optimisations?

`String.toUpperCase()`

Used in case-insensitive String comparisons.

439 bytes in 91 lines of code

Domain character set is limited?

Could a specialised version be faster?

hotspot/src/cpu/x86/vm/c2_globals_x86.hpp

```
// Sets the default values for platform dependent flags used by the server compiler.
// (see c2_globals.hpp).  Alpha-sorted.
define_pd_global(bool, BackgroundCompilation,      true);
define_pd_global(bool, UseTLAB,                    true);
define_pd_global(bool, ResizeTLAB,                 true);
define_pd_global(bool, CIPCompileOSR,              true);
define_pd_global(bool, InlineIntrinsics,           true);
define_pd_global(bool, PreferInterpreterNativeStubs, false);
define_pd_global(bool, ProfileTraps,                true);
define_pd_global(bool, UseOnStackReplacement,      true);
#ifdef CC_INTERP
define_pd_global(bool, ProfileInterpreter,         false);
#else
define_pd_global(bool, ProfileInterpreter,         true);
#endif // CC_INTERP
define_pd_global(bool, TieredCompilation,          trueInTiered);
define_pd_global(intx, CompileThreshold,           10000);
define_pd_global(intx, BackEdgeThreshold,          100000);

define_pd_global(intx, OnStackReplacePercentage,   140);
define_pd_global(intx, ConditionalMoveLimit,       3);
define_pd_global(intx, FLOATPRESSURE,              6);
define_pd_global(intx, FreqInlineSize,             325);
define_pd_global(intx, MinJumpTableSize,           10);
#ifdef AMD64
define_pd_global(intx, INTPRESSURE,                 13);
define_pd_global(intx, InteriorEntryAlignment,      16);
define_pd_global(intx, NewSizeThreadIncrease, ScaleForWordSize(4*K));
define_pd_global(intx, LoopUnrollLimit,             60);
// InitialCodeCacheSize derived from specjbb2000 run.
define_pd_global(intx, InitialCodeCacheSize,       2496*K); // Integral multiple of CodeCacheExpansionSize
define_pd_global(intx, CodeCacheExpansionSize,     64*K);

// Ergonomics related flags
define_pd_global(uint64_t, MaxRAM,                  128ULL*G);
#else
```

TL;DR

- Eliminate other performance issues first
- Keep your methods small for inlining
- Add a JIT log check step to your build process?
 - JITWatch suggestion tool
 - “hot method too big”
 - Unpredictable branches
- Learn about the JVM :)

Premature optimization is the root of all evil

Donald Knuth

Resources

- JITWatch on GitHub
 - <http://www.github.com/AdoptOpenJDK/jitwatch>
 - AdoptOpenJDK project
 - Pull requests are very welcome!
- Mailing list
 - groups.google.com/jitwatch
- Twitter
 - @chriswhocodes

Thanks!